commodore

The Transactor

comments and bulletins concerning your COMMODORE PET tm

Vol. 2
BULLETIN # 3

PET tm is a registered trademark of Commodore Business Machines Inc.

July 31,'79

When J. Butterfield's memory map was published in Transactor 9, Vol. 1, the first page of the ROM map section was somehow omitted. Therefore it has been included in this Transactor along with Jim's most recent memory map of the upgrade ROMs.

Introducing: The WEDGE

The WEDGE, or more commonly known as the "greater than" key (>), is likely to become the standard control key for adding functions to BASIC. These extra functions, of course in machine language, will be called by a preceeding ">". The program following was written by B. Seiler in California. Note the simulated "SAVE" in lines 100 to 220.

Append = bijwegen, toewegen

ATTENTION: Transactor Subscribers

Complete this coupon and take it to your local PET dealer to obtain a FREE 28 x 22, full colour PET Poster. But hurry...supplies are limited.

Name _____

Address _____

Date of Purchase ___/___/___ PET Model _____

Serial Number _____

Commodore Systems. 3370 Pharmacy Avenue, Agincourt, Ont.(416)499-4292

THE APPEND WEDGE (For 8Ks only) by B. Seiler

The APPEND WEDGE is an excellent program to append one BASIC Program to another. This special characteristic allows you to have a set of general purpose subroutines and 'tack' them onto any program. One draw-back though; the line numbers must be in order.

Because the edification of this listing is quite complete, you may wish to read through the listing first, and then commence to programming.

ENTERING THE "APPEND WEDGE"

1. First load the MACHINE LANGUAGE MONITOR Commodore Part No.321000
2. Use the MONITOR to enter the machine code into the second cassette buffer. Hex Ø33A to Ø3FF
3. Use the "X" command to return to BASIC . Type "NEW" (return).
4. Enter the BASIC PROGRAM. Lines 1Ø thru 23Ø are all that are necessary to LOAD, RUN, and SAVE "APPEND WEDGE".
5. Basic Lines 1ØØØ thru 9ØØØ are just the instructions.
6. To SAVE the original copy type RUN 1ØØ. This will save the Machine Language along with the BASIC Program.

```
10 SYS826:NEW
20 REM ************************
30 REM *                      *
40 REM * TO SAVE TYPE RUN 100 *
50 REM *                      *
60 REM ************************
100 POKE241,1
110 POKE247,58:POKE248,3
120 B=PEEK(124):POKE229,B
130 B=PEEK(125):POKE230,B
140 REM *** FIND SAVE NAME ***
150 A$=""
160 A$=STR$(PEEK(150)+256*PEEK(151))
170 A=VAL(A$)
180 A$="APPEND WEDGE"
190 B=PEEK(A):POKE238,B
200 B=PEEK(A+1):POKE249,B
210 B=PEEK(A+2):POKE250,B
220 SYS63153
230 END
```

```
1000 REM ********************************
1010 REM *                              *
1020 REM * FOR INSTRUCTIONS RUN 1000 *
1030 REM *                              *
1040 REM ********************************
1050 PRINT"■";
1100 PRINT"       ■ APPEND  WEDGE   COMMAND "
1110 PRINT:PRINT:PRINT
1120 PRINT"    THIS PROGRAM ADDS AN EXTRA COMMAND
1130 PRINT"TO PET BASIC.  THE EXTRA COMMAND IS
1140 PRINT"IS CALLED ■APPEND■.  ■APPEND■ ALLOWS THE
1150 PRINT"USER TO JOIN SEPERATE BASIC PROGRAMS.
1160 PRINT"■APPEND■ COULD BE USED TO LINK TESTED
1170 PRINT"SUBROUTINES TO A NEW MAIN PROGRAM.
1180 PRINT"    THE ■APPEND■ COMMAND IS ADDED TO
1190 PRINT"BASIC BY PLACING A WEDGE IN THE ZERO-
1200 PRINT"PAGE CODE USED TO SCAN ALL LINES.
1210 PRINT"THE WEDGE IS FORCED BY LINE 10 AND THE
1220 PRINT"PROGRAM AREA IS CLEARED BY A NEW.
1230 PRINT"    THE MACHINE CODE FOR ■APPEND■ SITS
1240 PRINT"IN THE SECOND CASSETTE BUFFER.  THIS
1250 PRINT"BUFFER IS FROM 033A HEX TO 0400 HEX OR
1260 PRINT"JUST BEFORE THE BASIC SOURCE.  TO SAVE
1270 PRINT"■APPEND■ THE SECOND CASSETTE BUFFER
1280 PRINT"MUST BE SAVED WITH THE BASIC SOURCE.
1300 GOSUB9000
1320 PRINT"■■■■    THE BASIC LINES 100 TO 230 PERFORM
1330 PRINT"THE TOTAL SAVE.  LINE 100 SETS THE
1340 PRINT"FIRST ADDRESS FOR CASSETTE #1.  LINE
1350 PRINT"110 SETS THE LO AND HI BYTES FOR THE
1360 PRINT"START ADDRESS OF THE SAVE TO 033A HEX.
1370 PRINT"LINES 120 AND 130 SET THE END ADDRESS
1380 PRINT"FOR THE SAVE TO VARTAB.  VARTAB POINTS
1390 PRINT"TO THE END OF BASIC SOURCE.
1400 PRINT"    A SPECIAL TRICK IS USED TO MAKE THE
1410 PRINT"NAME FOR THE SAVE.  LINES 140 THRU 170
1420 PRINT"LOCATE THE LENGTH AND ADDERSS POINTER
1430 PRINT"USED BY BASIC FOR STRING A$.
1440 PRINT"LINE 180 MAKES A$ EQUAL TO THE NAME FOR
```

```
 1450 PRINT"THE SAVE.  LINE 190 SETS THE LENGTH OF
 1460 PRINT"A$ FOR THE SAVE.  LINES 200 AND 210 SET
 1470 PRINT"THE ADDRESS OF A$ FOR THE SAVE NAME.
 1480 PRINT"FINALLY LINE 220 CALLS THE OPERATING
 1490 PRINT"SYSTEM ROUTINE TO DO THE SAVE.
 1500 GOSUB9000
 1600 PRINT"■■■■    TO ACTIVATE THE ■APPEND■ COMMAND
 1610 PRINT"TYPE ■RUN■.
 1620 PRINT"■    TO SAVE THE ■APPEND■ COMMAND AND
 1630 PRINT"INSTRUCTIONS TYPE ■RUN 100■.
 1640 PRINT"■■■■ WARNING "
 1650 PRINT"■    THE ■APPEND■ COMMAND DOES NOT FIX
 1660 PRINT"LINE NUMBERS!  APPENDING PROGRAMS WITH
 1670 PRINT"LINE NUMBERS OUT OF ORDER WILL HAVE
 1680 PRINT"STRANGE RESULTS WHEN RUN.
 1690 PRINT"    USE PET RENUMBER TO FIX SEGMENTS
 1700 PRINT"BEFORE APPENDING.
 1900 GOSUB9000
 2000 PRINT"■SYNTAX FOR ■APPEND■ COMMAND
 2010 PRINT
 2020 PRINT">APPEND "CHR$(34)"PROGRAM NAME"CHR$(34)
 2030 PRINT"↑  ↑      ↑
 2040 PRINT"|  |      |┌──────────────────────────┐
 2050 PRINT"|  |      └┤ NAME OF PROGRAM ON TAPE  |
 2060 PRINT"|  |       | #1 YOU WISH TO APPEND    |
 2070 PRINT"|  |       | TO THE PRESENT PROGRAM   |
 2080 PRINT"|  |       | IN PET MEMORY            |
 2090 PRINT"|  |       |                          |
 2100 PRINT"|  |       | IF OMITTED THE NEXT      |
 2110 PRINT"|  |       | PROGRAM ON TAPE #1 WILL  |
 2120 PRINT"|  |       | BE APPENDED              |
 2130 PRINT"|  |       └──────────────────────────┘
 2140 PRINT"|  |
 2150 PRINT"|  |┌──────────────────────────┐
 2160 PRINT"|  └┤ COMMAND NAME - A FOR SHORT |
 2170 PRINT"|   └──────────────────────────┘
 2180 PRINT"|┌──────────────────────────┐
 2190 PRINT"└┤ PROMPT CHARACTER MUST BE   |
 2200 PRINT" | IN FIRST COLUMN OF LINE    |
 2210 PRINT" └──────────────────────────┘
 2900 GOSUB9000
 5000 GOTO1000
 9000 PRINT"■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■";
 9010 PRINT"        ■ HIT ANY KEY TO CONTINUE ■";
 9020 GETA$:IFA$=""THEN9020
 9030 RETURN
READY.
```

```
LINE # LOC      CODE        LINE

0002  0000                ;*********************************
0003  0000                ;*
0004  0000                ;*  APPEND BASIC PROGRAMS
0005  0000                ;*
0006  0000                ;*    FOR LEVEL 1 BASIC
0007  0000                ;*
0008  0000                ;*    3-29-79
0009  0000                ;*
0010  0000                ;*
0011  0000                ;*********************************


0013  0000                ;BASIC VARABLES
0014  0000                TXTPTR=$C9
0015  0000                BUF=$0A
0016  0000                VARTAB=$7C


0018  0000                ;BASIC ROUTINES
0019  0000                CHRGET=$00C2          ;INC TXTPTR AND GETS CHAR
0020  0000                CHRGOT=$00C8          ;GETS LAST CHAR
0021  0000                FINI=$C430            ;FIXES LINKS


0023  0000                ;OP SYSTEM VARABLES
0024  0000                TEMP1=$50
0025  0000                EAL=$E5               ;END ADR FOR SAVE
0026  0000                EAH=$E6
0027  0000                FNLEN=$EE             ;LENGTH OF FILE NAME
0028  0000                FA=$F1
0029  0000                STAL=$F7              ;START ADR FOR SAVE
0030  0000                STAH=$F8
0031  0000                FNADR=$F9             ;ADDRESS OF FILE NAME
0032  0000                VERCK=$020B
0033  0000                SATUS=$020C
0034  0000                TAPE1=$027A
0035  0000                TAPE2=$033A


0037  0000                ;OP SYSTEM ROUTINES
0038  0000                FAH=$F5AE             ;READ ANY TAPE HEADER
0039  0000                LDAD2=$F64D           ;COPY PTRS FROM HEADER
0040  0000                SAVE=$F6B1            ;OP SYSTEM TAPE SAVE
0041  0000                PRT=$FFD2             ;PRINT CHAR IN A
0042  0000                LD410=$F42B           ;PRINT 'ING' MSG
0043  0000                TRD=$F88A             ;READ DATA FROM TAPE
0044  0000                TWAIT=$F913           ;WAIT FOR KEYBRD IRQ
0045  0000                LERR=$F3DB            ;PRT ERROR MSG
0046  0000                LD210=$F3E5           ;PRT READY,FIX VARTAB
0047  0000                ZZZ=$F667             ;SETS TBUF PTR
0048  0000                CSTE1=$F83B           ;ISSUE TAPE MSG
```

```
LINE # LOC      CODE        LINE

0049  0000                  LD300=$F3FF             ;PRINTS FILE NAME
0050  0000                  FAF=$F495               ;SEARCHS FOR FILE BY NAME
0051  0000                  OP160=$F579             ;PRT 'FILE NOT FOUND ERROR'



0053  0000                        *=TAPE2
0054  033A                  ;SET WEDGE CMD IN Z-PAGE CODE
0055  033A                  ;
0056  033A  A9 4C          SETW    LDA #$4C        ;JMP INSTRUCTION
0057  033C  85 CB                  STA CHRGOT+3
0058  033E  A9 47                  LDA #<WEDGE     ;SET LO
0059  0340  85 CC                  STA CHRGOT+4
0060  0342  A9 03                  LDA #>WEDGE     ;SET HI
0061  0344  85 CD                  STA CHRGOT+5
0062  0346  60              STRTS   RTS



0064  0347                  ;APPEND WEDGE CMD
0065  0347                  ;
0066  0347  C9 3E          WEDGE   CMP #'>         ;A WEDGE CMD?
0067  0349  D0 08                  BNE WG200       ;NO
0068  034B  48                    PHA             ;MAYBE
0069  034C  A5 C9                  LDA TXTPTR      ;WAS '>' IN COLUMN 1
0070  034E  C9 0A                  CMP #<BUF
0071  0350  F0 08                  BEQ WCMD        ;YES-WEDGE CMD



0073  0352  68              WG100   PLA             ;FINISH CHRGOT
0074  0353  C9 3A          WG200   CMP #':
0075  0355  B0 EF                  BCS STRTS
0076  0357  4C CF 00               JMP CHRGOT+7



0078  035A  20 C2 00      WCMD    JSR CHRGET      ;AN APPEND CMD?
0079  035D  C9 41                  CMP #'A
0080  035F  D0 F1                  BNE WG100       ;NO



0082  0361  A2 01                  LDX #1          ;YES-SET FOR CASSETTE #1
0083  0363  86 F1                  STX FA
0084  0365  CA                    DEX             ;X=$00
0085  0366  86 EE                  STX FNLEN       ;ZERO LENGTH OF FILE NAM
0086  0368  86 FA                  STX FNADR+1     ;POINT INTO Z-PAGE
0087  036A  8E 0B 02               STX VERCK       ;SET FOR A LOAD
```

```
LINE # LOC      CODE        LINE

0089  036D  20 C2 00    WC100  JSR CHRGET       ;GET NEXT CHAR
0090  0370  AA                 TAX              ;IS THIS THE END
0091  0371  F0 17               BEQ WC210        ;YES-LOAD ANYTHING
0092  0373  C9 22               CMP #$22         ;A (")?
0093  0375  D0 F6               BNE WC100        ;NO-LOOP
0094  0377  A6 C9               LDX TXTPTR       ;START FILE NAME ONE+
0095  0379  E8                  INX
0096  037A  86 F9               STX FNADR


0098  037C  20 C2 00    WC200  JSR CHRGET       ;FIND END OF THE NAME
0099  037F  AA                 TAX              ;THIS THE END?
0100  0380  F0 08               BEQ WC210        ;YES
0101  0382  C9 22               CMP #$22         ;AN END DOUBLE QUOTE?
0102  0384  F0 04               BEQ WC210        ;YES
0103  0386  E6 EE               INC FNLEN        ;NO-KEEP CHARACTER
0104  0388  D0 F2               BNE WC200        ;BRANCH ALWAYS


0106  038A  20 67 F6    WC210  JSR ZZZ          ;SET TBUF PTRS
0107  038D  20 3B F8           JSR CSTE1        ;ISSUE TAPE MSG
0108  0390  20 FF F3           JSR LD300        ;PRT FILE NAME
0109  0393  A5 EE               LDA FNLEN        ;LOADING ANY FILE
0110  0395  F0 08               BEQ WC250        ;YES
0111  0397  20 95 F4           JSR FAF          ;NO-SEARCH FOR IT
0112  039A  D0 08               BNE WC270        ;SKIP IF FOUND
0113  039C  4C 79 F5    WC220  JMP OP160        ;FILE NOT FOUND MSG
0114  039F  20 AE F5    WC250  JSR FAH          ;READ ANY FILE
0115  03A2  F0 F8               BEQ WC220        ;ERROR IF NOT FOUND


0117  03A4                    ;CALC NEW ADDRESS FOR APPEND SOURCE
0118  03A4                    ;
0119  03A4  AD 7D 02    WC270  LDA TAPE1+3      ;GET LO END ADR
0120  03A7  38                 SEC              ;CALC DELTA
0121  03A8  ED 7B 02           SBC TAPE1+1      ;SUB BEGIN LO
0122  03AB  AA                 TAX              ;SAVE IN X
0123  03AC  AD 7E 02           LDA TAPE1+4      ;GET END ADR HI
0124  03AF  ED 7C 02           SBC TAPE1+2      ;SUB BEGIN ADR HI
0125  03B2  A8                 TAY              ;SAVE IN Y
0126  03B3  A5 7C               LDA VARTAB       ;CALC APPEND BEGIN
0127  03B5  38                 SEC
0128  03B6  E9 04               SBC #4
0129  03B8  8D 7B 02           STA TAPE1+1      ;SET LOAD NEW START
0130  03BB  A5 7D               LDA VARTAB+1
0131  03BD  E9 00               SBC #0
0132  03BF  8D 7C 02           STA TAPE1+2      ;NEW START HI
0133  03C2  8A                 TXA              ;GET DELTA LO
0134  03C3  18                 CLC              ;CALC NEW LOAD END ADR
0135  03C4  6D 7B 02           ADC TAPE1+1
0136  03C7  8D 7D 02           STA TAPE1+3
0137  03CA  98                 TYA              ;GET DELTA HI
```

```
LINE # LOC      CODE        LINE

0138  03CB  6D 7C 02            ADC TAPE1+2
0139  03CE  8D 7E 02            STA TAPE1+4


0141  03D1  20 4D F6            JSR LDAD2       ;COPY POINTERS FOR LOAD
0142  03D4  A2 00               LDX #0          ;PRT 'APPENDING' MSG
0143  03D6  8E 0B 02            STX VERCK       ;CLEAR FOR A LOAD
0144  03D9  BD F8 03  WC300     LDA MSG1,X      ;DONE WITH MSG
0145  03DC  F0 06               BEQ WC400       ;YES
0146  03DE  20 D2 FF            JSR PRT
0147  03E1  E8                  INX
0148  03E2  D0 F5               BNE WC300       ;BRANCH ALWAYS


0150  03E4  20 2E F4  WC400     JSR LD410+3     ;PRINT 'ING' MSG
0151  03E7  20 8A F8            JSR TRD         ;READ DATA FROM TAPE
0152  03EA  20 13 F9            JSR TWAIT       ;WAIT FOR KEY IRQ RESTORE
0153  03ED  AD 0C 02            LDA SATUS       ;GOOD LOAD?
0154  03F0  F0 03               BEQ WC500       ;YES
0155  03F2  4C DB F3            JMP LERR        ;NO-LOAD ERROR


0157  03F5  4C E5 F3  WC500     JMP LD210       ;GO FIX BASIC LINKS
0158  03F8  0D        MSG1      .BYTE $D,'APPEND',0
0158  03F9  41 50
0158  03FF  00
0159  0400            AAAA
0160  0400                      .END


ERRORS = 0000


SYMBOL TABLE

SYMBOL    VALUE

AAAA      0400    BUF       000A    CHRGET    00C2    CHRGOT    00C8
CSTE1     F83B    EAH       00E6    EAL       00E5    FA        00F1
FAF       F495    FAH       F5AE    FINI      C430    FNADR     00F9
FNLEN     00EE    LD210     F3E5    LD300     F3FF    LD410     F42B
LDAD2     F64D    LERR      F3DB    MSG1      03F8    OP160     F579
PRT       FFD2    SATUS     020C    SAVE      F6B1    SETW      033A
STAH      00F8    STAL      00F7    STRTS     0346    TAPE1     027A
TAPE2     033A    TEMP1     0050    TRD       F88A    TWAIT     F913
TXTPTR    00C9    VARTAB    007C    VERCK     020B    WC100     036D
WC200     037C    WC210     038A    WC220     039C    WC250     039F
WC270     03A4    WC300     03D9    WC400     03E4    WC500     03F5
WCMD      035A    WEDGE     0347    WG100     0352    WG200     0353
ZZZ       F667
END OF ASSEMBLY
```

PET Variable Storage

Most PET users have, at one time or another, checked FRE(0) immediately after a LOAD and again after RUNning and found the two don't match. The difference can be minimal or sometimes quite drastic...but why ? The reason (as you have probably already guessed) is variable storage.

In PET BASIC there are generally three types of variables: string, floating point and integer (array variables are handled differently than these and will be discussed later). When PET executes statements such as:

```
A = 4.8                (direct)
10 LET Y = 17.5
20 X% = 1
30 B$ = "XXXX"
```

...it stores these variables and their assignments in variable storage space; RAM memory space determined by pointers in PET's operating system which are set up on power up or after a LOAD, as the case may be. They are stored in the order in which they are encountered.

Let's take a look at how each of these variables are handled by PET. First you'll need the machine language monitor. 8k users will have to LOAD the monitor and follow these 6 preliminary steps:

1. LIST. 10 SYS(1039) should appear. If not, record the "SYS" number.

2. RUN the monitor

3. Type exactly 'M 007A,0097' and hit RETURN. The following should appear:

```
            0  1  2  3  4  5  6  7
.:  007A 01 04 6B 07 6B 07 6B 07
.:  0082 00 20 0C 21 00 20 0A 00
.:  008A 98 0E 00 04 04 08 00 04
.:  0092 CC 0C 8C EA 24 C6 88 80
.
```

4. Now take the cursor up and change the following bytes (hit 'RETURN' after each line):

```
            0  1  2  3  4  5  6  7
.:  007A 01 08 03 08 03 08 03 08
.:  0082 .. .. .. .. .. .. .. ..
.:  008A .. .. .. .. .. .. .. ..
.:  0092 .. .. .. .. 04 08 .. ..
.
```

5. Type 'M 0800,0807' and RETURN. The following should appear:

```
            0  1  2  3  4  5  6  7
.:  0800 24 24 24 24 24 24 24 24
.
```

6. Change to:

```
            0  1  2  3  4  5  6  7
.:  0800 00 00 00 24 24 24 24 24
.
```

PET has now been fooled into thinking that BASIC memory space now starts at 0800 instead of 0400. This protects the machine language monitor from being clobbered when extra BASIC is entered.

NOTE: Of course all this is unnecessary for 16/32k users as the M.L.M. is in ROM and can't be trampled on by anything. Now, however, any further instructions will require one set of addresses for 8k's and another for 16/32k's as the results of this exercise will end up in different areas of memory for the two machines. Therefore, the 16/32k user will use the addresses or parameters placed in brackets.

Exit the monitor and enter and RUN the following BASIC:

```
20 A  = 2.5          <
30 B% = 9            <No spaces
40 C$ = "XXX"        <
```

After RUNning, re-enter the monitor with SYS1039 (SYS4 for 16/32k). Then do the following memory display:

```
.:  M 0800,0840   (0400,0440)
.:        0  1  2  3  4  5  6  7
.:  0800 00 0B 08 14 00 41 B2 32
.:  0808 2E 35 00 14 08 1E 00 42
.:  0810 25 B2 38 00 21 08 28 00
.:  0818 43 24 B2 22 58 58 58 22
.:  0820 00 00 00 41 00 82 20 00
.:  0828 00 00 C2 80 00 09 00 00
.:  0830 00 43 80 03 1C 08 00 00
.:  0838 24 24 24 24 24.......
```

Figure 1.

16k users will of course see "04's" rather than "08's" and the "24's" at the bottom will be "AA's" indicating "empty space".

Notice the first 4 rows is our BASIC program followed by three "00's" indicating 'end of BASIC'. Following this is the variable table which we'll get into right now.

Floating Point Variables

Floating point implies a numeric value with a fractional component. In our case it will be A = 2.5 . This value is stored along with its corresponding variable in the 7 memory locations following 0823 (0423) inclusive:

9

0823 41 00 82 20 00 00 00

Variations of the above 7 locations is all that is required to store any floating point number within the upper and lower limits of PETs floating point range.

The first two bytes are used to store the variable. 41 is "A" in hexadecimal. The next byte is set aside for double character variables (e.g. AA=2.5). Since ours is only a single character, location 0824 will be 00 as shown.

The remaining 5 bytes are for the actual value itself. The 82 is the exponent of the value. This is offset by 80 (half of FF) such that negative exponents can also be obtained. In our case 2 is added to indicate that the decimal point is 2 places to the right of the most signifigant bit. As you know, binary 2 = ...0*4 + 1*2 + 0*1....:

```
... 2  2  2  2  2  2 . 2  2  2  2  2  2 ...

    0  0  0  0  1  0 . x  x  x  x  x  x
```

That covers the integer part...now the fractional part. We have 2 so far. We need to represent .5 more. Therefore a "1" is required in the 2 column. This is contained in the next byte following the exponent. 0826 contains 20 which, in binary, is:

0 0 1 0 0 0 0 0

This is "OR'd" into the above such that the leftmost bit is beneath the most signifigant bit of the integer part of the number:

```
        ... 2  2  2  2  2  2 . 2  2  2  2  2  2 ...

            0  0  0  0  1  0 . x  x  x  x  x  x
OR'd with                  0  0   1  0  0  0  0  0
        ___________________________________________
        =   0  0  0  0  1  0 . 1  0  0  0  0  0
```

...which is 1*2 + 1*.5 = 2.5 !

Lastly is the sign of the value. If you study the theory of this method of deriving numbers, you'll notice that the leftmost bit of the "OR'd with" number never has to be a 1 for determining the magnitude of the number. Therefore it is used as the sign bit and is set to 1 for negative numbers. Examples of this and more floating point numbers at the end of this article.

Integer Variables

Integers are those with no fractional component and are stored by PET in a much simpler fashion. In our case, B% is stored in the 7 bytes immediately following A . But how does PET know that this variable is any different from the last. Notice the first two bytes of B% as compared to A :

```
0823 41 00
082A C2 80 00 09 00 00 00
```

Since A is represented as 41 in hex, you might expect that B is 42. Well you're right; B is 42 in hex but when B (or any other letter) is employed as an integer variable, bit 8 is set to 1 such that PET can make the distinction. Looking at the table on the last page of the last Transactor, you'll see that the letters stop at decimal 90 and therefore never use the 8th bit. Expanding...

```
"A"  = 41 = 0 1 0 0  0 0 0 1
"B"  = 42 = 0 1 0 0  0 0 1 0
"B%" = C2 = 1 1 0 0  0 0 1 0
```

Bit 8 of the second byte of an integer variable is also set even if a double variable name is not used.

The next two bytes of the seven are the only ones used to represent the value. The remaining three are never used. Integers take no less space than floating point except in arrays. This simplifies the search process.

The first byte used in representing the value, 082C (042C), is the high order byte and the second 082D (042D), is the low order. The two are of course the hex representation of the value in decimal. Recall that the maximium integer value possible is 32767 or 8000 hex. The remaining possibilities are used for negative integers. Some examples:

```
B% =      9 = 00 09
B% =    256 = 01 00
B% =    257 = 01 01
B% =      0 = 00 00
B% =     -1 = FF FF
B% =   -256 = FE 00
B% = -32767 = 80 01
```

String Variables

For every string variable created, another 7 bytes are used up by PET but of course the string itself is not stored there. Our string variable, C$, is stored beginning at 0831 (0431). PET distinguishes string variables by setting bit 8 over the second byte only. "C" is 43 in hex:

```
0831 43 80 03 1C 08 00 00
```

Location 0833 (0433) holds the length of the string (Recall...40 C$ = "XXX"). The following two bytes are the low and high order bytes of the address of the string. Inotherwords, why store the string again when it already exists in the text area. Instead simply store a pointer which points at the first character of the string and call up X number of characters following where X equals the 'length' byte (03 in our case).

This procedure is fine for strings which are defined in text, but what about those that are not. Take for example the following:

```
100 INPUT " YOUR NAME ";A$
200 D$ = RIGHT$ ( A$ , 1 )
300 C$ = D$ + "*" + A$    .
```

In cases like these, PET stores the strings at the end of available RAM moving down and creates a pointer in the variable table to the beginning of the string.

The Search Process

Each time a variable is defined, 7 bytes of memory are used up. When a variable is called by BASIC in lines such as:

```
400 IF A = 1 THEN A = A + 5
500 PRINT B% , C$          .
600 ON A GOTO 1020 , 1030  .
700 X = X - 3              .
```

...PET starts at the beginning of the variable table, determined by the pointer at 007C & 007D (002A, 002B), and examines the first pair of bytes. If an exact match is not made, PET jumps 7 locations to the next variable. The search continues until the variable is found and if not found is assumed to be zero or null for strings.

Once established, PET loads the value or string into a work area and performs the desired operation. In a situation such as line 700, PET must find X (zero or otherwise), load it into the accumulator, find X again, subtract 3 and re-asign X. Of course all this takes time and if X resides down at the end of the table, PET must scan through all the variables ahead of X before it finds X. Therefore, if a variable is known to be used more often than others, time can be saved by "setting up" the variable table at the beginning of the program:

```
10 X = 0 : A$ = " " : B% = 0 : Y = 0
```

This can speed things up considerably especially if X is called upon each pass of a long FOR-NEXT loop.

What You Can Do

Assuming you still have the monitor running with the display as in Figure 1, change the following (do not exit the monitor):

```
.:  M 0800,0840   (0400,0440)
.:        0  1  2  3  4  5  6  7
.:  0800 .. .. .. .. .. .. .. ..
.:  0808 .. .. .. .. .. .. .. ..
.:  0810 .. .. .. .. .. .. .. ..
.:  0818 .. .. .. 22 59 59 59 22
.:  0820 .. .. .. .. .. 83 20 ..
.:  0828 .. .. .. .. .. 0F .. ..
.:  0830 .. .. .. 05 1C 07 .. ..
.:  0838 .. .. .. .. .. .. .. ..
.
```

Now type "X" and RETURN to exit the monitor and execute the following line directly on the screen:

```
? A , B% , C$
```

A is now 5 because the exponent of A was incremented by 1. This means that everything was shifted left one position putting the most significant bit (MSB) in the 2^2 column and Least significant bit (LSB) in the 2^0 column. 1*4 + 1*1 = 5 .

B% equals 15 now since the low order byte of B% was changed to 0F.

If you've ever tried programming this, you know it's impossible:

```
40 C$ = ""YYY""
```

PET interprets this as null string followed by the variable 'YYY' followed by null string. But now C$ prints out as '""YYY""' because the address of the string was changed as well as the length.

Floating Point Examples

The magnitude of a floating point value is always stored in 5 bytes. The other two are reserved for the variable name and will be ignored here so that we can concentrate on the format of the value.

Floating point is handled by PET in this format ('M' = Mantissa):

```
EXP   M1   M2   M3   M4   SIGN
```

The sign is contained in M1 but is "extracted" on its way into the accumulator and placed in a 'sign register'.

Ex 1. EXP M1 M2 M3 M4
85 22 40 00 00

Since the EXP is 85, the decimal point will be 5 positions to the right of the MSB (Most Significant Bit):

EXP = _ _ _ 1 0 0 0 0 . 0 0 _ _ _

So far the magnitude is 16.

M1 = 22 = 0 0 1 0 0 0 1 0

M2 = 40 = 0 1 0 0 0 0 0 0

M3 = M4 = 0

To complete the operation, M1 and M2 are concatenated...

M1 + M2 = 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0

...and OR'd with the EXP such that the leftmost bit of M1 + M2 is under the MSB of the value:

EXP = _ _ _ 1 0 0 0 0 . 0 0 _ _

OR'd with: M1 + M2 = _ _ _ 0 0 1 0 0 . 0 1 0 0 1 0 0 0 0 0 0

Equals: _ _ _ 1 0 1 0 0 . 0 1 0 0 1 0 _ _ _ _ _

This is still the binary representation. The decimal value is now:

$1*2^{4} + 0*2^{3} + 1*2^{2} + 0*2^{1} + 0*2^{0} + 0*2^{-1} + 1*2^{-2} + 0*2^{-3} + 0*2^{-4} + 1*2^{-5} + 0*2^{-6} + \ldots$

..which equals...

1*16 + 1*4 + 1*.25 + 1*.03125 = 20.28125

EXP M1 M2 M3 M4
Therefore 20.28125 = 85 22 40 00 00

Ex 2. EXP M1 M2 M3 M4
8A FF E7 80 00

Since the EXP is 8A, the decimal point will be 10 positions to the right of the MSB.

EXP = _ _ _ 1 0 0 0 0 0 0 0 0 0 . 0 0 _ _

Notice that bit 8 of Mantissa 1 is set. Therefore, the sign of the value will be negative. Now M1, M2, M3 and M4 must be concatenated:

M1 = FF = 1111 1111
M2 = E7 = 1110 0111
M3 = 80 = 1000 0000
M4 = 00

M(1+2+3) = 1111 1111 1110 0111 1000 0000

...and OR'd with the EXP...

EXP = _ _ 1 0 0 0 0 0 0 0 0 0 . 0 0

OR: M = _ _ 1 1 1 1 1 1 1 1 1 1 . 1 0 0 1 1 1 1 0 0 0 0 0

Equals = _ _ 1 1 1 1 1 1 1 1 1 1 . 1 0 0 1 1 1 1 0 _ _ _ _

..which equals...:

$2^9 + 2^8 + 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 + 2^{-1} + 2^{-4} + 2^{-5} + 2^{-6} + 2^{-7}$

In decimal:

512 + 256 + 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 + .5 + .0625 + .03125 + .015625 + .0078125 = 1023.6171875

However, the sign is negative...therefore:

	EXP	M1	M2	M3	M4
-1023.6171875 =	8A	FF	E7	80	00

Ex 3. EXP M1 M2 M3 M4
7E E0 00 00 00

The EXP is less than 80 indicating the result will be less than 1. Now the decimal point will be 2 positions to the left of the MSB because 7E is 2 less than 80:

EXP = _ _ _ 0 0 . 0 1 0 0 _ _ _ _ _ _

M1 = E0 = 1110 0000
M2 = 00
M3 = 00
M4 = 00

EXP = _ _ . 0 1 0 _ _ _ _ _ _
OR: M = _ _ . _ 1 1 1 0 0 0 0 0

Equals _ _ . 0 1 1 1 0 0 _ _ _

In decimal = -(.25 + .125 + .0625)
= -.4375

Nothing to it, right? Try these:

Ex 4. EXP M1 M2 M3 M4
84 48 00 00 00 = _ _ _ _ _ . _ _ _

Ex 5. EXP M1 M2 M3 M4
7F C0 00 00 00 = _ _ . _ _ _ _ _ _

Ex 6. EXP M1 M2 M3 M4
__ __ __ 00 00 = 2 9 . 5

Array variables next month!

IEEE BUS HANDSHAKE ROUTINE IN MACHINE LANGUAGE

To use the IEEE-488 bus on the PET at maximum speed it is necessary to use machine language rather than BASIC 'INPUT' and 'PRINT'. The routine given here has been used with an HP3437A systems voltmeter to reach data transfer speeds of over 5000 bytes per second, corresponding to 2500 voltage readings in 2-byte packed binary format or 625 readings in 8-byte ASCII format. The best speed attained in BASIC is 75 readings per second transferred as character strings.

The IEEE bus

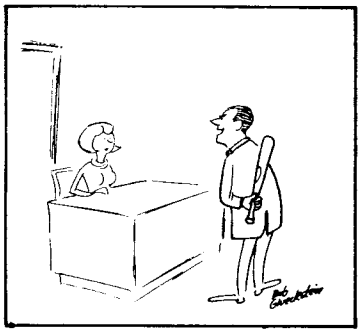
Details of the IEEE-488 bus are given in the PET Users Handbook, but some clarification of the register addresses on page 120 of the handbook is helpful. These are:

HEX	DECIMAL	BITS	IEEE	DIRECTION
E820	59424	0-7	DIO 1-8	from bus
E822	59426	0-7	DIO 1-8	to bus; 'PET'controlled
E821	59425	3	NDAC	'PET' controlled
E823	59427	3	DAV	'PET' controlled
E840	59456	0	NDAC	from bus
		1	NRFD	'PET' controlled
		2	ATN	'PET' controlled
		6	NRFD	from bus
		7	DAV	from bus

Note that on the IEEE bus, 'high' is logic false and 'low' is logic true; and that the data bus must be left with all bits 'high' when PET has finished to avoid confusion of data put on to the bus by other devices.

Cont'd ➡

****



'I'd Like to Reason With Your Computer.'

'A Sudden Reduction of Personnel is Indicated.'

Geplaatst

The program controls a given number of data transfers, each of 8 bytes, from the HP3437A to the PET. Each one is preceded by a trigger (GET - group execute trigger) on the IEEE bus, and the HP3437A must be correctly addressed as a 'talker' or a 'listener' at all times by sending MTA (my talk address) or MLA (my listen address) before transfers as appropriate. The sending of messages (GET, MTA, MLA, etc.) or data is controlled by the ATN line; ATN is true when messages are being sent.

The program and returned data are held in the top 2K of memory; this is hidden from BASIC using POKE 134,255 : POKE 135,23 as the first line of the BASIC control program. The number of readings required is POKEd into 6400_{10}, then control passed to the machine language program by SYS(6144). The data bytes coming in on the IEEE bus are stored in locations 6401_{10} onwards; these are PEEKed out on return to BASIC, and converted into numbers using the function VAL. As the index register is used for counting, only 256 bytes can be transferred using this program, but it would be easy to modify the program to perform more transfers.

Disassembled listings with comments and a separate listing (for ease of copying into BASIC DATA statements!) are given.

This program was prepared using a machine language handler written by the author, and the listings produced by this handler and by a modified version of the 'disassemble' part of the PETSOFT (C) ASSEMBLER 'EXEC' program.

IEEE bus handshake routine - main program

```
1800 A200   LDX #00     prepare index register
1802 A9FB   LDA #FB     set ATN low
1804 2D40E8 AND E840
1807 8D40E8 STA E840
180A A928   LDA #28     MLA (28 for this device)
180C 8501   STA 01
180E 208018 JSR 1880    handshake into bus
1811 A908   LDA #08     GET
1813 8501   STA 01
1815 208018 JSR 1880    handshake
1818 A948   LDA #48     MTA
181A 8501   STA 01
181C 208018 JSR 1880    handshake
181F A9FD   LDA #FD     set NRFD low (ready to receive data)
1821 2D40E8 AND E840
1824 8D40E8 STA E840
1827 A9F7   LDA #F7     and NDAC low also
1829 2D21E8 AND E821
182C 8D21E8 STA E821
182F A904   LDA #04     set ATN high
1831 0D40E8 ORA E840
```

```
1834 8D40E8 STA E840
1837 A008   LDY #08      ready to count 8 bytes
1839 20B018 JSR 18B0     handshake data from bus
183C A502   LDA 02       result to A
183E 9D0119 STA 1901,X   store in  1901+X
1841 E8     INX
1842 88     DEY
1843 D0F4   BNE 1839     jump if Y not zero
1845 A9FB   LDA #FB      set ATN low
1847 2D40E8 AND E840
184A 8D40E8 STA E840
184D A902   LDA #02      set NRFD high
184F 0D40E8 ORA E840
1852 8D40E8 STA E840
1855 A908   LDA #08      set NDAC high
1857 0D21E8 ORA E821
185A 8D21E8 STA E821
185D A95F   LDA #5F      UNT
185F 8501   STA 01
1861 208018 JSR 1880     handshake to bus
1864 A904   LDA #04      set ATN high
1866 0D40E8 ORA E840
1869 8D40E8 STA E840
186C CE0019 DEC 1900     decrease counter
186F D091   BNE 1802     jump if not zero
1871 60     RTS          return to BASIC program
```

subroutine to handle handshake into bus

```
1880 AD40E8 LDA E840     NRFD ?
1883 2940   AND #40
1885 F0F9   BEQ 1880     jump back if not ready
1887 A501   LDA 01       ready:  get data byte
1889 49FF   EOR #FF      complement it
188B 8D22E8 STA E822     send to bus
188E A9F7   LDA #F7      set DAV low
1890 2D23E8 AND E823
1893 8D23E8 STA E823
1896 AD40E8 LDA E840     NDAC ?
1899 2901   AND #01
189B F0F9   BEQ 1896     jump back if not accepted
189D A908   LDA #08      accepted;  set DAV high
189F 0D23E8 ORA E823
18A2 8D23E8 STA E823
18A5 A9FF   LDA #FF      255(10) into bus
18A7 8D22E8 STA E822
18AA 60     RTS          return to main
```

subroutine to handle handshake from bus

```
18B0 A902   LDA #02      set NRFD high
18B2 0D40E8 ORA E840
1885 8D40E8 STA E840
18B8 AD40E8 LDA E840     DAV ?
18BB 2980   AND #80
18BD D0F9   BNE 18B8     jump back if not valid
18BF AD20E8 LDA E820     get data byte from bus
18C2 49FF   EOR #FF      complement
18C4 8502   STA 02       store in $ 0002
```

```
18C8 2D40E8 AND E840
18CB 8D40E8 STA E840
18CE A908   LDA #08     set NDAC high
18D0 0D21E8 ORA E821
18D3 8D21E8 STA E821
18D6 AD40E8 LDA E840    DAV high ?
18D9 2980   AND #80
18DB F0F9   BEQ 18D6    jump back if not
18DD A9F7   LDA #F7     set NDAC low
18DF 2D21E8 AND E821
18E2 8D21E8 STA E821
18E5 A9FF   LDA #FF     255(10) into bus
18E7 8D22E8 STA E822
18EA 60     RTS         return to main
```

IEEE bus handshake routine listing

```
1800 A2 00 A9 FB 2D 40 E8 8D
1808 40 E8 A9 28 85 01 20 80
1810 18 A9 08 85 01 20 80 18
1818 A9 48 85 01 20 80 18 A9
1820 FD 2D 40 E8 8D 40 E8 A9
1828 F7 2D 21 E8 8D 21 E8 A9
1830 04 0D 40 E8 8D 40 E8 A0
1838 08 20 B0 18 A5 02 9D 01
1840 19 E8 88 D0 F4 A9 FB 2D
1848 40 E8 8D 40 E8 A9 02 0D
1850 40 E8 8D 40 E8 A9 08 0D
1858 21 E8 8D 21 E8 A9 5F 85
1860 01 20 80 18 A9 04 0D 40
1868 E8 8D 40 E8 CE 00 19 D0
1870 91 60 EA EA EA EA EA EA
1878 EA EA EA EA EA EA EA EA
1880 AD 40 E8 29 40 F0 F9 A5
1888 01 49 FF 8D 22 E8 A9 F7
1890 2D 23 E8 8D 23 E8 AD 40
1898 E8 29 01 F0 F9 A9 08 0D
18A0 23 E8 8D 23 E8 A9 FF 8D
18A8 22 E8 60 EA EA EA EA EA
18B0 A9 02 0D 40 E8 8D 40 E8
18B8 AD 40 E8 29 80 D0 F9 AD
18C0 20 E8 49 FF 85 02 A9 FD
18C8 2D 40 E8 8D 40 E8 A9 08
18D0 0D 21 E8 8D 21 E8 AD 40
18D8 E8 29 80 F0 F9 A9 F7 2D
18E0 21 E8 8D 21 E8 A9 FF 8D
18E8 22 E8 60
```

```
0001 data to go into bus
0002 data from bus

1900 counter for number of data transfers

1901 start of results area
```

John A. Cooke

Department of Astronomy
University of Edinburgh
Royal Observatory
Edinburgh EH9 3HJ

A few routines from PET basic — Compiled by Jim Butterfield, Toronto

(Original ROM)

C2AC-C2D9 peeks at the stack for an active FOR loop
C2DA-C31C 'opens up' a space in Basic for insertion of a new linex.
C31D-C329 tests for stack-too-deep and aborts if found.
C32A-C356 check available memory space
C357-C388 sends a canned error message from C190 area, then drops into:
C389-C391 Signals 'ready'
C394-C3A9 gets a line of input, analyzes it, executes it
C3AC-C42E handles a new line of Basic from keyboard; deletes old line, etc.
C430-C460 corrects the chaining between Basic lines after insert/delete
C462-C476 receives a line from the keyboard into the Basic buffer
C479-C48C gets each character from keyboard
C48D-C521 looks up the keywords in an input lines and changes to "tokens"
C522-C550 searches for the location of a Basic line from number in 8,9
C551-C599 implements NEW command - clears everything (011/019 ROM change)
C59A-C5A7 sets the Basic pointer to start-of-program
C5A8-C647 performs LIST command
C649- C68F executes a FOR statement
C692-C6B4 continues to build FOR vectors
C6B5-C6EF reads and executes the next Basic statement, finds next line, etc.
C6F2-C70A executes the Basic Command as a subroutine
C70D-C71B performs RESTORE
C71C-C742 handles STOP, END, and BREAK procedures.
C745-C75E performs CONT
C75F-C76D set pause after carriage return (never called)
C770-C772 performs CLR
C775-C77D performs RUN
C780-C79A performs GOSUB
C79D-C7C9 performs GOTO
C7CA-C7FD performs RETURN
C7FE-C81E scans for start of next Basic Line
C820-C840 performs IF
C843-C862 performs ON
C863-C89A gets a fixed-point number from Basic and stores in 8,9
C89D-C91B performs LET
C91C-C97E check numeric digit/move string pointer
C97F-C982 performs PRINT#
C985-C996 performs CMD
C999-CA24 performs PRINT
CA27-CA41 prints string from address in Y,A
CA44-CA76 prints a character
CA77-CA9E handles bad input data
CA9F-CAC5 performs GET
CAC6-CADF performs INPUT#
CAE0-CB14 performs INPUT
CB17-CB21 prompts and receives the input
CB24-CC11 performs READ
CC12-CC35 canned messages: EXTRA IGNORED; REDO FROM START
CC36-CC8F performs NEXT
CC92-CCB5 checks Basic format, data type, flags TYPE MISMATCH
CCB8-CD38 inputs and evaluates any expression (numeric or string)
CD3a-CD9C pushes a partially-evaluated argument to the stack
CD9D-CDB9 evalues a numeric, variable, or pi, or identifies other symbol
CDBC-CDC0 value of pi in floating binary

Routines in Upgrade ROM Jim Butterfield, Toronto

C000-C045 Action addresses for primary keywords
C046-C073 Action addresses for functions
C074-C091 Hierarchy and action addresses for operators
C092-C192 Table of Basic keywords
C193-C2A9 Basic messages, mostly error messages.
C2AA-C2D7 Search stack for FOR or GOSUB activity
C2D8-C31A Open up space in memory
C31B-C327 Test: stack too deep?
C328-C354 Check available memory
C355 Send canned error message, then:
C389-C3AA Print READY.
C3AB-C441 Handle new Basic line from keyboard
C442-C46E Rebuild chaining of Basic lines in memory.
C46F-C494 Receive line from keyboard
C495-C52B Change keywords to Basic tokens
C52C-C55A Search Basic for a given Basic line number
C55B Perform NEW, then:
C577-C5A6 Perform CLR
C5A7-C5B4 Reset Basic execution to start-of-program
C5B5-C657 Perform LIST
C658-C6FF Perform FOR
C700-C72F Execute Basic statement
C730-C73E Perform RESTORE
C73F-C76A Perform STOP and END
C76B-C784 Perform CONT
C785-C78F Perform RUN
C790-C7AC Perform GOSUB
C7AD-C7D9 Perform GOTO
C7DA Perform RETURN, and perhaps:
C7F3-C80D Perform DATA, i.e., skip rest of statement
C80E Scan for next Basic statement
C811-C82F Scan for next Basic line
C830 Perform IF, and perhaps:
C843-C852 Perform REM, i.e., skip rest of line
C853-C872 Perform ON
C873-C8AC Get fixed-point number from Basic
C8AD-C927 Perform LET
C928-C936 Add ASCII digit to accumulator #1
C937-C98A Continue to perform LET
C98B-C990 Perform PRINT#
C991-C9A4 Perform CMD
C9A5-CA1B Perform PRINT
CA1C-CA38 Print string from memory
CA39-CA4E Print single format character (space, cursor-right, ?)
CA4F-CA7C Handle bad input data
CA7D-CAA6 Perform GET
CAA7-CAC0 Perform INPUT#
CAC1-CAF9 Perform INPUT
CAFA-CB06 Prompt and receive input
CB07-CBFB Perform READ; common routines used by INPUT and GET
CBFC-CC1F Messages: EXTRA IGNORED, REDO FROM START
CC20-CC78 Perform NEXT
CC79-CC9E Checks data type, prints TYPE MISMATCH
CC9F Inputs & evaluates any expression (numeric or string)

CDEC Evaluate expression within parentheses ()
CDF2 Check right parenthesis)
CDF5 Check left parenthesis (
CDF8-CE02 Check for comma
CE03-CE07 Print SYNTAX ERROR and exit
CE08-CE0E Set up function for future evaluation
CE0F-CE88 Search for variable name
CE89-CEC7 Identify and set up function references
CEC8 Perform OR
CECB-CEF7 Perform AND
CEF8-CF5F Perform comparisons, string or numeric
CF60-CF6C Perform DIM
CF6D-CFF6 Search for variable location in memory
CFF7-D000 Check if ASCII character is alphabetic
D001-D077 Create new Basic variable
D078-D088 Array pointer subroutine
D089-D08C 32768 in floating binary
D08D-D0AB Evaluate expression for positive integer
D0AC-D227 Find or create array
D228-D258 Compute array subscript size
D259 Perform FRE
D26D-D279 Converts fixed-point to floating-point
D27A-D27F Perform POS
D280-D28C Check if direct command, print ILLEGAL DIRECT
D28D-D2BA Perform DEF
D2BB-D2CD Check FNx syntax
D2CE-D33E Evaluate FNx
D33F-D34E Perform STR$
D34F-D360 Calculate string vector
D361-D3CD Scan and set up string
D3CE-D3FF Subroutine to build string vector
D400-D496 Garbage collection subroutine
D497-D4DF Check for most eligible string for collection
D4E0-D516 Collect a string
D517-D553 Perform string concatenation
D554-D57C Build string into memory
D57D-D5B4 Discard unwanted string
D5B5-D5C5 Clean the descriptor stack
D5C6-D5D9 Perform CHR$
D5DA-D605 Perform LEFT$
D606-D610 Perform RIGHT$
D611-D63A Perform MID$
D63B-D655 Pull string function parameters from stack
D656-D65B Perform LEN
D65C-D664 Move from string-mode to numeric-mode
D665-D674 Perform ASC
D675-D686 Input byte parameter
D687-D6C5 Perform VAL
D6C6-D6D1 Get two parameters for POKE or WAIT
D6D2-D6E7 Convert floating-point to fixed-point
D6E8-D706 Perform PEEK
D707-D70F Perform POKE
D710-D72B Perform WAIT
D72C-D732 Add 0.5 to accumulator#1
D733-D744 Perform subtraction
D745-D76D Microsoft joke

D76E-D852 Perform addition
D853-D889 Complement accumulator#1
D88A-D88E Print OVERFLOW and exit
D88F-D8C7 Multiply-a-byte subroutine
D8C8-D8F5 Function constants: 1, SQR(.5), SQR(2), -0.5, etc.
D8F6 Perform LOG
D937-D964 Perform multiplication
D965-D997 Multiply-a-bit subroutine
D998-D9C2 Load accumulator #2 from memory
D9C3-D9DF Test and adjust accumulators #1 and #2
D9E0-D9ED Handle overflow and underflow
D9EE-DA04 Multiply by 10
DA05-DA09 10 in floating binary
DA0A Divide by 10
DA13 Perform divide-into
DA1E-DAAD Perform divide-by
DAAE-DAD2 Load Accumulator #1 from memory
DAD3-DB07 Store accumulator #1 into memory
DB08-DB17 Copy accumulator #2 into accumulator #1
DB18-DB26 Copy accumulator #1 into accumulator #2
DB27-DB36 Round off accumulator #1
DB37-DB44 Compute SGN value of accumulator #1
DB45-DB63 Perform SGN
DB64-DB66 Perform ABS
DB67-DBA6 Compare accumulator #1 to memory
DBA7-DBD7 Convert floating-point to fixed-point
DBD8-DBFE Perform INT
DBFF-DC89 Convert string to floating-point
DC8A-DCBE Get new ASCII digit
DCBF-DCCD String conversion constants: 99999999, 999999999, 1E+9
DCCE Print IN, followed by:
DCD9-DCE8 Print Basic line number
DCE9-DE1C Convert number or TI$ to ASCII
DE1D-DE5D Constants for numeric conversion
DE5E Perform SQR
DE68 Perform power function
DEA1-DEAB Perform negation
DEAC-DED9 Constants for string evaluation
DEDA-DF2C Perform EXP
DF2D-DF76 Function series evaluation subroutines
DF77-DF7E Manipulation constants for RND
DF7F-DFD7 Perform RND
DFD8 Perform COS
DFDF-E027 Perform SIN
E028-E053 Perform TAN
E054-E08B Constants for trig evaluation: pi/2, 2*pi, .25, etc.
E08C-E0BB Perform ATN
E0BC-E0F8 Constants for ATN series evaluation
E0F9-E110 Subroutine to be moved to zero page ($70 to $87)
E111-E115 Initial RND seed
E116-E1B6 Initialize Basic system
E1B7-E1DD Messages: BYTES FREE, ### COMMODORE BASIC ###
E1DE Initialize I/O registers, and:
E229 Clear screen, and:
E257-E284 Home cursor

E285-E2F3 Input from screen or keyboard; wait for input completion
E2F4-E33E Input from screen
E33F-E34B Test for quotation mark and reverse quote-flag
E34C-E38A Set up screen print parameters
E38B-E395 Prevent 80-character line from getting any longer
E396-E3B3 Extend 40-character line to 80 characters
E3B4-E3D7 Back into the previous line (via DEL or CURSOR LEFT key)
E3D8-E518 Handle ASCII character for screen output
E519-E53E Go to next line on screen
E53F-E5B9 Scroll the screen
E5BA-E61A Open a line on the screen (via INSERT key)
E61B-E62D Main interrupt entry point
E62E-E6E9 Hardware interrupt: service clock, keyboard, cassettes
E6EA-E6F7 Print character on screen
E6F8-E769 Table: decoder for keyboard matrix
E76A-E796 MLM subroutine: output hex digits
E797-E7A6 MLM subroutine: swap TMP0 and TMP2
E7A7-E7F6 MLM subroutine: input hex digits
E7F7-E7FF MLM subroutine: print ?
F000-F0B5 Monitor messages, mostly for Input/Output
F0B6 Set up IEEE for Listen, Talk, etc.
F0EE-F127 Send character to IEEE-488 bus
F128-F135 Output character immediate mode to IEEE-488
F136-F155 Send errors: WRITE TIMEOUT, DEVICE NOT PRESENT, etc.
F156-F163 Send canned I/O message
F164-F16E Send immediate Listen command, then secondary address
F16F-F17E Output character deferred mode to IEEE-488
F17F-F18B Drop IEEE channel: send Unlisten or Untalk
F18C-F1D0 Input character from IEEE-488 bus
F1D1-F1E0 GET a character
F1E1-F231 INPUT from any device
F232-F26D OUTPUT a character to any device
F26E Abort all files, and;
F284-F28C Restore normal I/O devices
F28D-F2A8 Find file table entry; set parameters from file table
F2A9-F300 Perform CLOSE
F301-F30E Test stop key
F30F-F314 Action stop key
F315-F31C Send message if direct mode
F31D-F321 Test if direct mode
F322-F3C1 Perform program loading
F3C2-F409 Perform LOAD
F40A-F43D Subroutines: Print SEARCHING... ; Print LOADING or VERIFYING
F43E-F45F Get Load or Save parameters
F460-F465 Get a byte parameter
F466-F493 Send program name to IEEE-488 bus
F494-F4B6 Find a specific tape header
F4B7-F4CD Perform VERIFY
F4CE-F50D Get parameters for OPEN, CLOSE
F50E-F515 Abort calling subroutine if end-of-line (default parameters)
F516-F520 Confirm comma, else send SYNTAX ERROR
F521-F5A5 Perform OPEN
F5A6-F5D9 Find any tape header
F5DA-F63B Write tape header

F63C-F655 Get start & end program addresses from tape header
F656-F66B Set cassette buffer address according to device number
F66C-F683 Set tape start & end addresses from buffer address
F684-F68C Perform CMD
F68D-F69D Set tape start & end addresses from Basic pointers
F69E-F728 Perform SAVE
F729-F76C Update TI and TI$, and copy STOP key to work area
F76D-F76F TI constant: limit of clock (24 hours)
F770-F7BB Set input device
F7BC-F805 Set output device
F806-F811 Advance tape buffer pointer (for INPUT#, GET#, and PRINT#)
F812-F834 Wait: PRESS PLAY ON TAPE#
F835-F846 Test if cassette button(s) pressed
F847-F854 Wait: PRESS PLAY & RECORD ON TAPE#
F855 Initiate tape read
F886-F8E5 Initiate tape write
F8E6-F8EF Test for I/O interrupt completion
F8F0-F8FF Test stop key
F900-F930 Set expected timing for next input bit from tape
F931 Interrupt entry: Read tape bits
FA57-FB75 Store received tape characters
FB76-FB7E Set tape read/write address back to starting point
FB7F-FB83 Flag I/O error into ST
FB84-FB92 Reset 8-count and flags for a new byte
FB93-FBAE Write a transition to cassette tape
FBAF-FC40 Write interrupt 2: write data to tape
FC41-FC7A Write interrupt 1: write tape shorts (leader)
FC7B-FC95 Terminate tape: restore normal interrupt vector
FC96-FCA5 Set interrupt vector from table
FCA6-FCB3 Turn off cassette motors
FCB4-FCC5 Perform running checksum calculation
FCC6-FCD0 Advance read/write pointer
FCD1-FCFD Power-on reset entry point
FCFE-FD00 NMI interrupt entry point
FD01-FD10 Table of interrupt vectors
FD11-FFB0 Machine Language Monitor (MLM) - see Commodore documentation
FFB1-FFBF CBM copyright statement
*****Jump Table*****
FFC0 OPEN
FFC3 CLOSE
FFC6 Set input device
FFC9 Set output device
FFCC Restore default I/O devices
FFCF Input character
FFD2 Output character
FFD5 LOAD
FFD8 SAVE
FFDB VERIFY
FFDE SYS
FFE1 Test stop key
FFE4 Get character
FFE7 Abort all I/O activity
FFEA Clock update
FFF0-FFF9 unused
FFFA-FFFF Hardware vectors: NMI, Reset, Interrupt

COMMODORE BUSINESS MACHINES LIMITED
3370 PHARMACY AVENUE, AGINCOURT, ONTARIO M1W 2K4
TELEPHONE (416) 499-4292 – CABLE ADDRESS: COMTYPE
TELEX NUMBER 06-525400